

Datenbanken und Datensicherheit

Übungsaufgaben 11 – 8. Januar 2025

Aufgabe 1: Verschlüsselung brechen

Brechen Sie die folgenden Verschlüsselungen und rekonstruieren Sie die (deutschsprachigen) Klartexte:

(a) Cäsar-Chiffre: `mpzjolyz mypag mpzjoa mypzjol mpzjol`

(b) Monoalphabetische Substitution: `pmn yvnkpn png inagfcna mgt vaoatogtjok`

Aufgabe 2: Bedienfehler bei Verschlüsselung

Der symmetrische Verschlüsselungsalgorithmus RC4, eine Stromverschlüsselung, galt lange Zeit als sicher, gilt aber seit 2013 als gebrochen. Bereits vorher gab es Probleme mit verbreiteter falscher Anwendung des Algorithmus, die die Verschlüsselung aushebelte.

Diese Aufgabe illustriert, wie *jede* Stromverschlüsselung durch falsche Anwendung unsicher wird.

RC4 zeichnet sich vor allem durch seine Einfachheit aus. Mit genügend Erfahrung im Programmieren ist es nicht schwer, allein aus dem Gedächtnis eine RC4-Software zu schreiben, die unter Verwendung des Protokolls CipherSaber-2 (siehe: <https://en.wikipedia.org/wiki/CipherSaber>) Daten ver- und entschlüsseln kann. Dies kann z. B. in Ländern relevant sein, die Zugang zu Verschlüsselungs-Software einschränken.

Die Datei `cs2.pl` (Perl, 5 Zeilen) enthält eine voll funktionsfähige Implementation von CipherSaber-2. (Eine längere, dafür lesbarere Implementation (C, 92 Zeilen) finden Sie in `cs2-pg.c`.)

Gebrauchsanleitung zu `cs2.pl`:

- Das Programm liest von der Standardeingabe und schreibt zur Standardausgabe.
- Das Programm macht eine `xor`-Verschlüsselung mit einem Strom von Pseudozufallszahlen. Es ist daher nicht nötig, zwischen Ver- und Entschlüsseln zu unterscheiden.
- Option `-k`: Schlüssel („key“)
- Option `-i`: Initialisierungsvektor

Der Initialisierungsvektor ist öffentlich und wird der verschlüsselten Nachricht vorangestellt. Der Schlüssel ist geheim. Ohne den Initialisierungsvektor – oder: bei gleichbleibendem Initialisierungsvektor – wird bei jeder Verwendung desselben Schlüssels dieselbe Folge von Pseudozufallszahlen erzeugt. Mit Hilfe von `xor`-Operationen ist es dann leicht möglich, wenn man zu *einem* verschlüsselten Text den Klartext kennt, auch *andere* Texte zu entschlüsseln, die mit *demselben* Schlüssel und Initialisierungsvektor verschlüsselt wurden, auch ohne den Schlüssel zu kennen.

Wir nutzen nun `cs2.pl`, um den Text „Dies ist ein nicht ganz so kurzer Test.“ zu verschlüsseln und wieder zu entschlüsseln:

```
$ echo "Dies ist ein nicht ganz so kurzer Test." \  
| ./cs2.pl -k="1234" -i="cvhtestcvh" > ciphertext-1.bin  
$ dd if=ciphertext-1.bin bs=1 skip=10 status=none \  
| ./cs2.pl -k="1234" -i="cvhtestcvh"  
cvhtestcvhDies ist ein nicht ganz so kurzer Test.
```

(Die Datei `ciphertext-1.bin` finden Sie in unserem GitLab.)

Jemand hat nun – unter Mißachtung der Gebrauchsanleitung – noch einen zweiten Text mit demselben Schlüssel `1234` und demselben Initialisierungsvektor `cvhtestcvh` verschlüsselt, siehe die Datei `ciphertext-2.bin`.

Aufgabe: Entschlüsseln Sie `ciphertext-2.bin`, ohne von der Kenntnis des Schlüssels (hier: `1234`) Gebrauch zu machen.

Viel Erfolg!